First person action networking game using virtual reality goggles in the Unity game engine

**Praca inżynierska   
 na kierunku Informatyka**

**Pierwszoosobowa sieciowa gra akcji dla gogli wirtualnej rzeczywistości**

**w silniku Unity**

**Kraków 2019**

**Politechnika Krakowska  
im. Tadeusza Kościuszki**

Wydział Fizyki, Matematyki i Informatyki

**Dawid Lubera**

Numer albumu: 117597

Praca wykonana pod kierunkiem:  
**dr inż. arch. Paweł Ozimek**

**Uzgodniona ocena:**......................................

...........................................................................podpisy promotora i recenzenta





SPIS TREŚCI

[1. WSTĘP 4](#_Toc534939129)

[1.1. CEL PRACY 4](#_Toc534939130)

[1.2. STRUKTURA PRACY 4](#_Toc534939131)

[2. WPROWADZENIE TEORETYCZNE 4](#_Toc534939132)

[2.1. KOMUNIKACJA W SIECI 4](#_Toc534939133)

[2.2. TECHNOLOGIE 4](#_Toc534939134)

[2.2.1. Unity 4](#_Toc534939135)

[2.2.2. Oculus Rift 5](#_Toc534939136)

[2.2.3. OpenVR 7](#_Toc534939137)

[2.2.4. C# 8](#_Toc534939138)

[2.2.5. GIT 8](#_Toc534939139)

[2.2.6. Visual Studio 8](#_Toc534939140)

[2.3. TECHNOLOGIE SIECIOWE W UNITY 9](#_Toc534939141)

[2.3.1. High Level API 9](#_Toc534939142)

[2.3.2. Low Level API 9](#_Toc534939143)

[2.3.3. Photon Unity Networking 10](#_Toc534939144)

[2.3.4. Wybór API 10](#_Toc534939145)

[2.4. TECHNOLOGIE WIRTUALNEJ RZECZYWISTOŚCI 10](#_Toc534939146)

[2.4.1. Hełm wideo 10](#_Toc534939147)

[2.4.2. Śledzenie pozycji 10](#_Toc534939148)

[2.4.3. Śledzenie ruchów głowy 10](#_Toc534939149)

[2.4.4. Śledzenie ruchów rąk 10](#_Toc534939150)

[2.4.5. Specyfikacja 11](#_Toc534939151)

[3. PRZEGLĄD ISTNIEJĄCYCH ROZWIĄZAŃ 11](#_Toc534939152)

[4. TWORZENIE APLIKACJI 11](#_Toc534939153)

[4.1. STWORZENIE PROJEKTU 11](#_Toc534939154)

[4.2. STWORZENIE MAPY 14](#_Toc534939155)

[4.3. KONFIGURACJA RUCHU DŁONI 18](#_Toc534939156)

[4.4. PORUSZANIE POSTACI 19](#_Toc534939157)

[4.4.1. Przemieszczanie modelu w przestrzeni 19](#_Toc534939158)

[4.4.2. Odwrotna kinematyka postaci 21](#_Toc534939159)

[4.5. KOMUNIKACJA SIECIOWA 24](#_Toc534939160)

[4.5.1. Synchronizacja ruchu postaci 24](#_Toc534939161)

[4.5.2. Stworzenie lobby 26](#_Toc534939162)

[4.6. SWORZENIE STANU GRY 26](#_Toc534939163)

[4.7. DODAWANIE DŹWIĘKÓW 26](#_Toc534939164)

[5. TESTOWANIE 26](#_Toc534939165)

[6. ZAKOŃCZENIE 27](#_Toc534939166)

[6.1. PODSUMOWANIE 27](#_Toc534939167)

[6.2. MOŻLIWOŚCI ROZWOJU 27](#_Toc534939168)

[7. PRZYPISY 28](#_Toc534939169)

[8. BIBLIOGRAFIA 29](#_Toc534939170)

[9. SPIS ILUSTRACJI 30](#_Toc534939171)

[DODATEK A. MODELE UŻYTE W PROJEKCIE 31](#_Toc534939172)

# WSTĘP

## 1.1. CEL PRACY

## 1.2. STRUKTURA PRACY

# WPROWADZENIE TEORETYCZNE

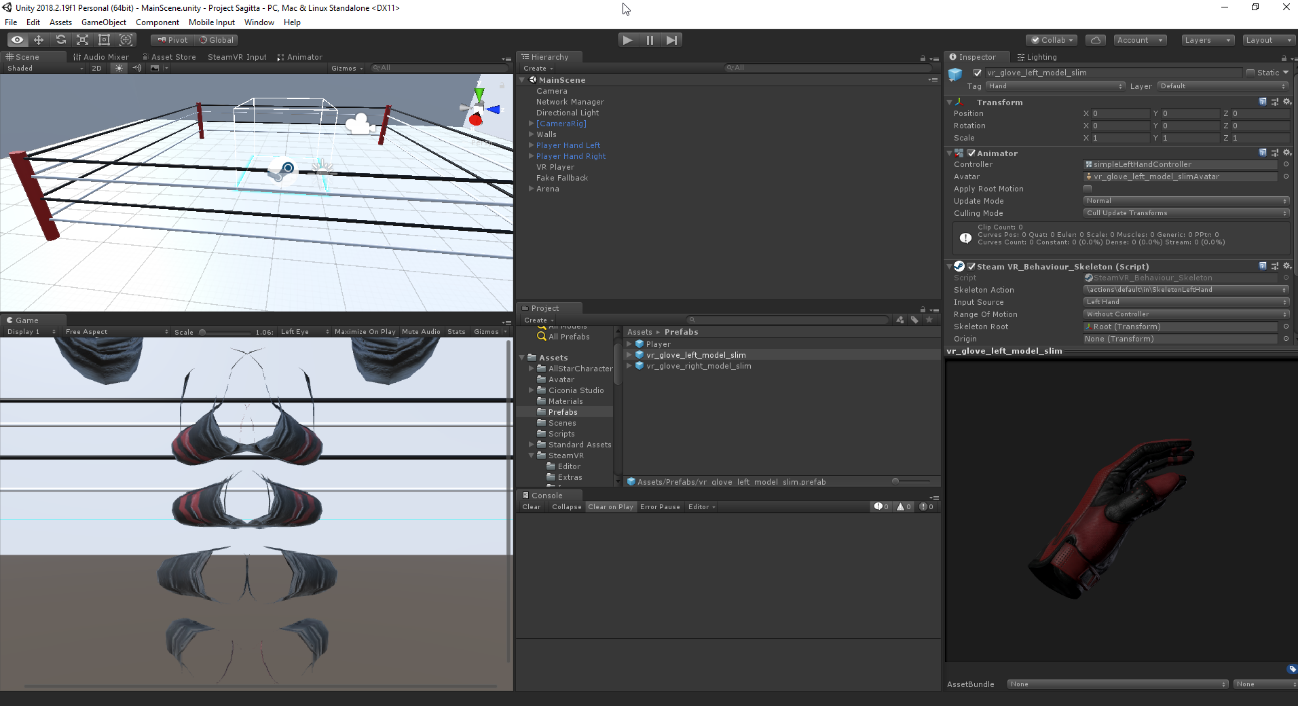
## KOMUNIKACJA W SIECI

## TECHNOLOGIE

### 2.2.1. Unity

Unity jest rozbudowanym silnikiem do gier, który pozwala na tworzenie, wyświetlanie i zarządzanie dwu i trójwymiarowymi światami. Darmowa wersja programu pozwala na korzystanie ze znacznej większości funkcji. Ograniczenia wiążą się z brakiem wsparcia producenta, ilości graczy w rozgrywkach multiplayer oraz maksymalny dochód ze sprzedaży po którym wymagana będzie płatna licencja.

W 2005 roku Unity zostało stworzone jako silnik do gier trójwymiarowych i wciąż jest to jego główne zadanie. Zawiera jednak wiele funkcji pozwalających w łatwy sposób tworzyć gry dwuwymiarowe. Środowisko to pozwala na kompilację gry na wiele różnych platform w tym: Windows, Linux, Mac OS X, PS4, Xbox One, Nintendo Switch, Android, iOS. Pozwala także tworzyć gry i aplikacje dla gogli wirtualnej rzeczywistości. W przypadku niektórych gogli wymagane jest pobranie odpowiednich zasobów ze sklepu Unity.



Rysunek 1. Główne okno edytora Unity

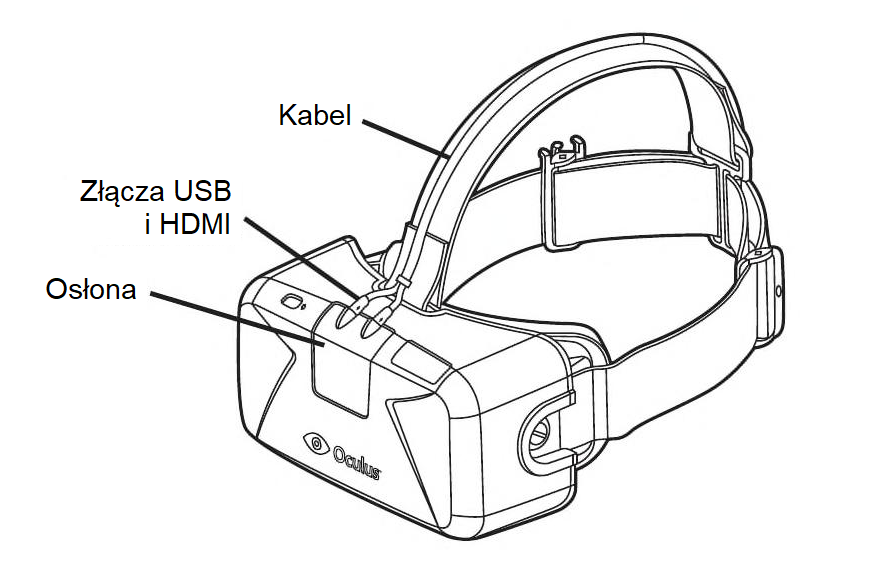
Edytor Unity pozwala na zaimportowanie do gry modeli w wielu popularnych formatach w tym z programów: 3D Studio Max, Maya, Blender, SketchUp, a także popularny w wielu innych programach format OBJ. Umożliwia import tekstur i materiałów bezpośrednio z plików modeli lub osobno w postaci plików graficznym między innymi w formatach JPEG i PNG.

Pracując w Unity część zadań można wykonać bez programowania. Tworzenie świata gry, proste łączenia elementów lub animacji nie wymagają pisania kodu. Te czynności można wykonać poprzez przeciąganie odpowiednich elementów, wybór odpowiednich trybów pracy czy zaznaczanie pól wyboru. Przy programowaniu używany jest język C#. Twórcy programu starają się na bieżąco wprowadzać nowe funkcjonalności języka do swojego silnika. Unity posiada także własny sklep, w którym można kupić gotowe skrypty, modele, pliki muzyczne czy narzędzia dodające nowe funkcjonalności do edytora.

### 2.2.2. Oculus Rift

Oculus Rift są pierwszymi nowoczesnymi goglami wirtualnej rzeczywistości. Pierwsza deweloperska została wydana w marcu 2013 roku, a pierwsza konsumencka wersja w styczniu 2016 roku. W grudniu tego samego roku zostały wydane kontrolery Touch.

Gogle wirtualnej rzeczywistości pozwalają na obserwowanie wirtualnego świata w sposób symulujący obecność w tym świecie. Wyświetlają one dwa obrazy, jeden dla każdego oka, które zasłaniają całe pole widzenia. Stosowane są odpowiednio zakrzywione soczewki oraz ruch kamery w grze lub filmie bazujący na faktycznych ruchach głowy. Sprawia to, że ludzki mózg faktycznie jest w stanie wczuć się w wydarzenia, które odbywają się na ekranach.



Rysunek 2. Schemat Oculus Rifta

Kontrolery Touch posiadają dokładne sensory pozwalające analizować pozycję, ruch i obrót dłoni użytkownika, a dokładne przyciski dające informację o sile nacisku. Kontrolery pozwalają na dokładne symulowanie dłoni i odwzorowanie ruchu każdego palca. Sprawienie, że gracz widzi ruchy swoich dłoni w grze sprawia, że immersja jest jeszcze głębsza. Na każdym z kontrolerów znajdują się dwa zwykłe przyciski pozwalające odczytywać stany czy jest wciśnięty - true czy nie - false, niewielki joytick kierunkowy, dotykowe miejsce obok przycisków, a także dwa triggery. Triggery w przeciwieństwie do przycisków pozwalają odczytać nie tylko informacje czy przycisk jest wciśnięty czy nie, ale także siłę nacisku – wartość od 0 do 1. Jeden trigger jest w pozycji palca wskazującego, a drugi, trzech palców: środkowego, serdecznego i małego. Każdy element zwraca także informację czy użytkownik go dotyka czy nie. Te wszystkie dane pozwalają stwierdzić w jaki sposób jest ułożona dłoń użytkownika.



Rysunek 3. Kontrolery Oculus Touch

### 2.2.3. OpenVR

OpenVR jest zestawem deweloperskim (SDK) stworzonym przez firmę Valve, aby ujednolicić tworzenie oprogramowania dla urządzeń wirtualnej rzeczywistości. Valve miało kluczowe znaczenie przy tworzeniu Oculus Rifta, a także stworzyło gogle HTC Vive i kontrolery dla dłoni Valve Knuckles. Wszystkie te urządzenia są wspierane przez OpenVR, jednak lista urządzeń, z których można używać z tym SDK jest znacznie dłuższa. Valve jest też twórcą dodatku do Unity dodającego wsparcie dla tego interfejsu.



Rysunek 4. Okno przypisywania przycisków do akcji

Gry stworzone przy pomocy OpenVR pozwalają także na bardzo łatwą zmianę zastosowań danych przycisków zarówno przez deweloperów jak i przez graczy. Pozwala także na analizę użycia przycisków w trakcie testowania aplikacji.

### 2.2.4. C#

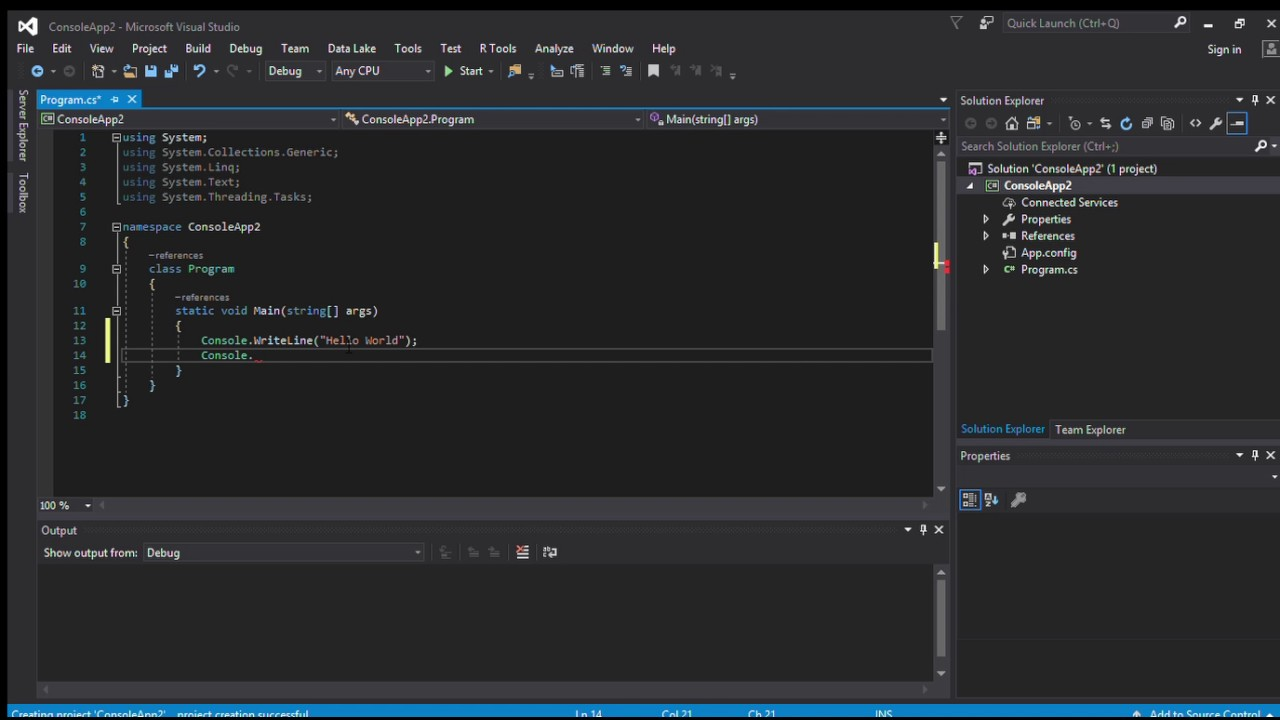
C# jest obiektowym językiem programowania stworzonym przez firmę Microsoft, którego pierwsza wersja ukazała się w lipcu 2000. Od tamtego czasu jest aktywnie rozwijany i wprowadzane są nowe funkcjonalności. Głównymi językami na których bazuje C# są C++, Java i Delphi. Kod programu napisanego w tym języku jest kompilowany do specjalnego języka Common Intermediate Language. Taki program może zostać uruchomiony przez środowisko uruchomieniowe np. .NET Framework, .NET Core lub Mono.

C# jest aktualnie jedynym językiem używanym przy programowaniu aplikacji w Unity.

### 2.2.5. GIT

### 2.2.6. Visual Studio

Visual Studio jest zintegrowanym środowiskiem programistycznym (IDE) stworzonym przez firmę Microsoft. Od kilku lat jest to podstawowe środowisko programistyczne używane z Unity w systemach Windows. Środowisko programistyczne pozwala na łatwiejsze zarządzanie kodem. Visual Studio wspiera programowanie w wielu popularnych językach programowania i kodowania między innymi: C, C++, C#, JavaScript, CSS czy HTML. Twórcy Visual Studio wprowadzają obsługę nowych funkcjonalności języków w nowych aktualizacjach.



Rysunek 5. Okno główne Visual Studio 2018

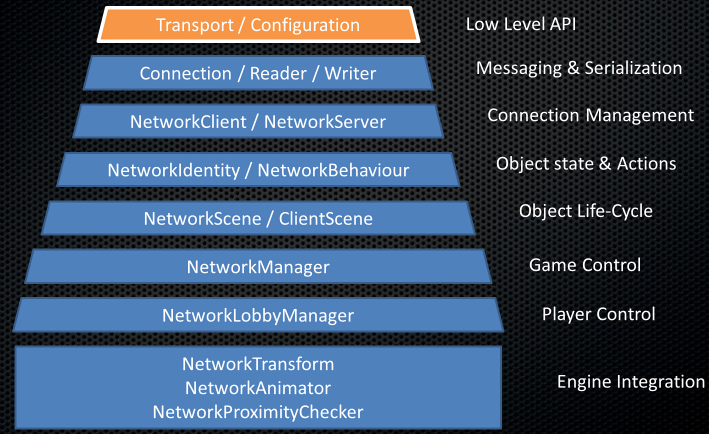
## TECHNOLOGIE SIECIOWE W UNITY

### 2.3.1. High Level API

High Level API (HLAPI) jest jednym z dwóch sieciowych API dostępnych w Unity. Jest ono zbudowane na bazie drugiego – Low Level API. High Level API udostępnia logikę wymaganą w grach sieciowych bez konieczności wchodzenia w szczegóły przesyłania danych w protokołach sieciowych. HLAPI pozwala także na łatwe włączenie matchmakingu czyli automatycznego parowania dwóch graczy, jeśli tylko są podłączeni do tego samego serwera. Darmowa wersja Unity pozwala na połączenie się do serwerów firmy Unity maksymalnie 20 graczy. W większości wypadków jest to bardzo wygodne i znacznie szybsze rozwiązanie. Ma ono jednak swoje wady, między innymi to, że wymusza w jaki sposób klient komunikuje się z serwerem oraz jak wiele danych i w jakim formacie przesyła klient.

### 2.3.2. Low Level API

Low Level API (LLAPI) zostało zaprojektowane w taki sposób, aby zapewnić znacznie większą elastyczność. Kosztem elastyczności tego rozwiązania jest jednak ilość pracy jaką należy wykonać do stworzenia systemu pozwalającego na połączenie i rozgrywkę dwóch klientów. LLAPI jest tylko nieco bardziej przystosowaną do obsługi gier nakładką na narzędzia działające w transportowej warstwie modelu TCP/IP.



Rysunek 6. Schemat warstwy sieciowej w Unity (stworzyć własny schemat)

Wybrane w projekcie HLAPI pozwala na częściowe używanie LLAPI, jednak w bardzo ograniczonym zakresie.

### 2.3.3. Photon Unity Networking

### 2.3.4. Wybór API

## TECHNOLOGIE WIRTUALNEJ RZECZYWISTOŚCI

### 2.4.1. Hełm wideo

### 2.4.2. Śledzenie pozycji

### 2.4.3. Śledzenie ruchów głowy

### 2.4.4. Śledzenie ruchów rąk

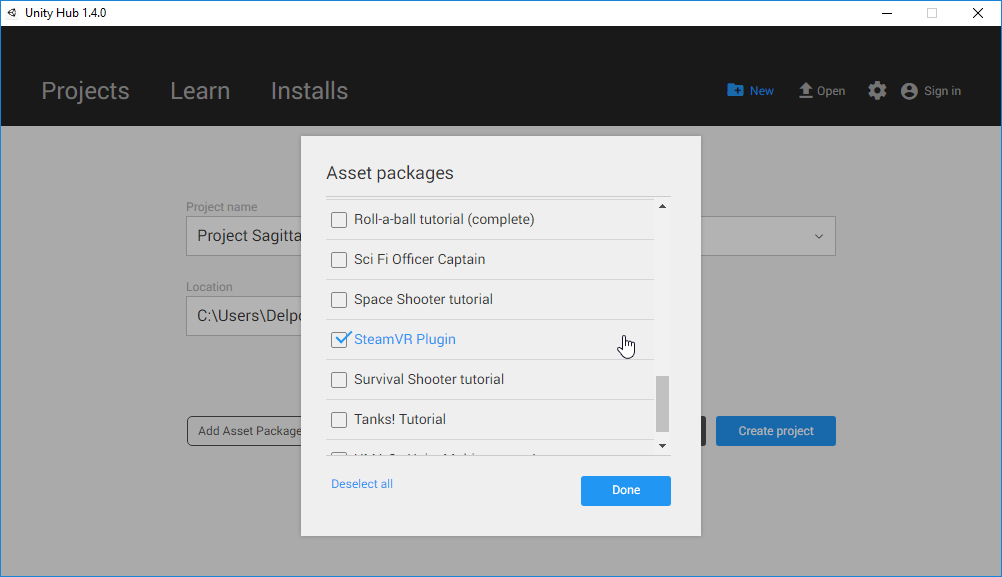
### 2.4.5. Specyfikacja

# PRZEGLĄD ISTNIEJĄCYCH ROZWIĄZAŃ

# TWORZENIE APLIKACJI

## 4.1. STWORZENIE PROJEKTU

Unity pozwala na łatwe stworzenie projektu klikając na przycisk *New* znajdujący się w prawym górnym rogu ekranu aplikacji startowej. W kolejnym kroku należy podać nazwę projektu, miejsce zapisu projektu w systemie plików oraz używany w projekcie szablon – 2D lub 3D. Okno to pozwala także na automatyczne dodanie dodatkowych zasobów (ang. asset) pobranych z Unity Asset Store.



Rysunek . Ekran tworzenia projektu w Unity

Projekt został skonfigurowany z następującymi opcjami:

* Nazwa projektu – Project Sagitta
* Szablon – 3D
* Lokalizacja – C:\Users\Delpod\Documents\Unity projects
* Dodatkowe zasoby – SteamVR Plugin

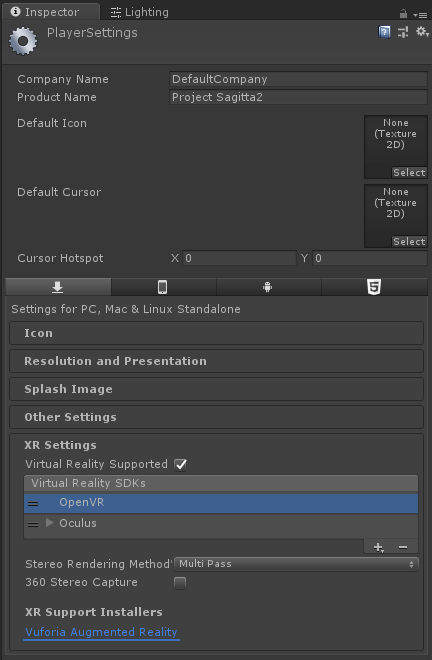
Po stworzeniu projektu przez Unity plugin SteamVR wyświetlił okno z zalecanymi ustawieniami konfiguracyjnymi projektu. Opcje które powinny zostać zmienione to:

* Wyświetlanie okna rozdzielczości – zmiana z wyświetlania na ukrycie. W przypadku aplikacji stworzonych na VR wyświetlanie powinno być unikane, ponieważ obraz wyświetla się na ekranach w hełmie i wymaga odpowiedniej rozdzielczości.
* Umożliwienie zmiany rozmiaru okna podglądu – nie ma znaczenia dla wyświetlania gry na goglach
* Przestrzeń kolorów – zmiana z Gamma na Linear z uwagi na mniejsze obciążenie karty graficznej



Rysunek 8. Okno konfiguracji opcji pluginu SteamVR

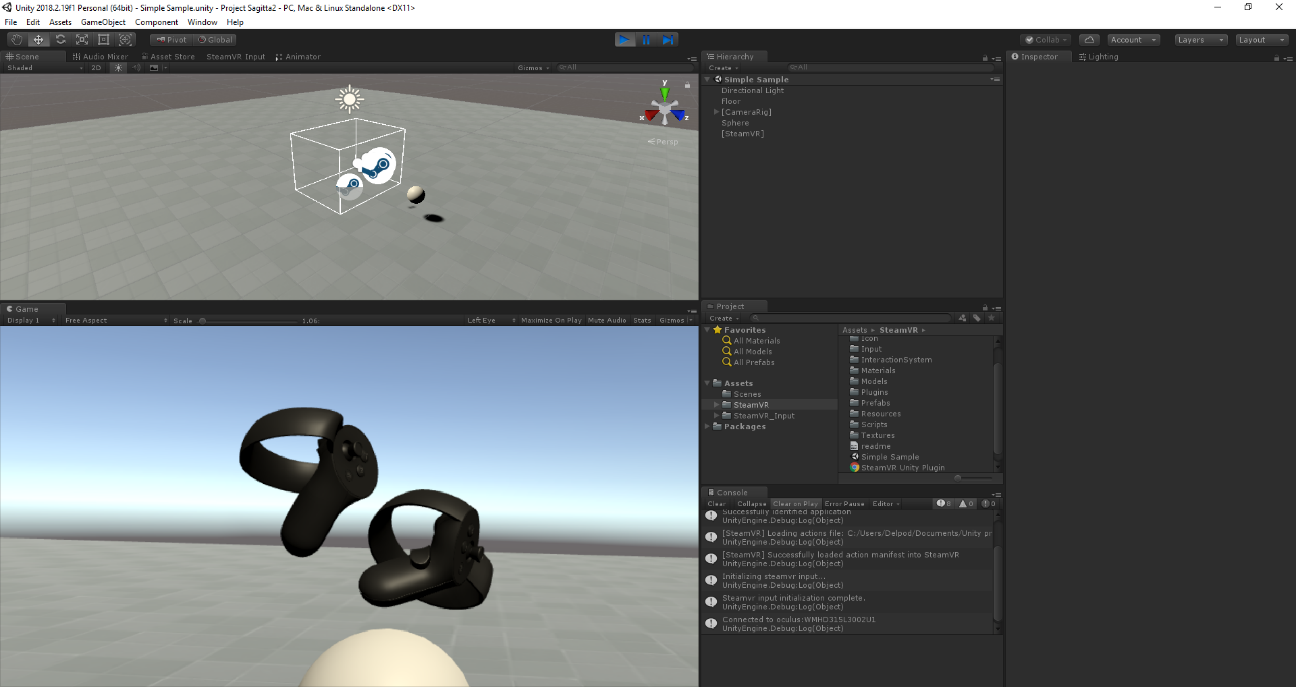
Kolejnym wymaganym krokiem jest zmiana ustawień projektu w Unity. Zostało to dokonane poprzez kliknięcie w oknie programu *Edit* > *Project Settings* > *Player*. W wyświetlanym oknie w sekcji XR zaznaczona została opcja *Virtual Reality Supported* oraz z uwagi na korzystanie z OpenVR zmieniona została kolejność SDK, tak, aby OpenVR znajdował się na pierwszym miejscu.



Rysunek 9. Konfiguracji Opcji XR

Do uruchomienia aplikacji wymagana była jeszcze scena, która zawiera kamerę oraz 2 obiekty odpowiadające każdej z dłoni. Plugin SteamVR zawiera taką podstawową scenę, więc została ona skopiowana do projektu i użyta jako baza do dalszej pracy.

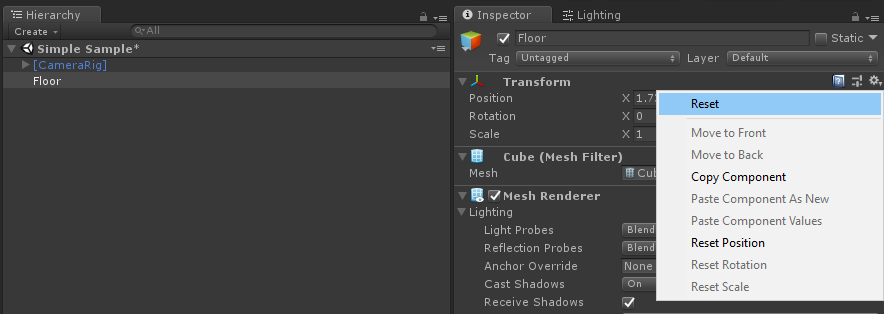
Uruchamiając projekt po raz pierwszy plugin SteamVR zaoferował możliwość konfiguracji wejścia. Podstawowe domyślne akcje zaproponowane przez plugin były wystarczające i wymagane było tylko ich wygenerowanie. Po ich zapisaniu można było już uruchomić projekt, który już na goglach wyświetlał podstawową scenę oraz umożliwiał poruszanie kontrolerami.



Rysunek 10. Okno edytora z uruchomioną podstawową sceną

## 4.2. STWORZENIE MAPY

Mapa miała składać się z oświetlonego pokoju z ringiem na środku. Na początku usunięta została zawartość sceny poza obiektem *[CameraRig]***.** Klikając prawym klawiszem myszy i wybierając *3D Object* > *Cube* w oknie hierarchii utworzony został nowy prostopadłościan. Następnie należało zresetować transformację tego obiektu klikając w ikonę zębatki komponentu *Transform* i wybierając *Reset*.



Rysunek . Resetowanie komponentu w Edytorze

Odpowiednio ustawiając zmienne w komponencie *Transform* utworzona została podłoga. Obiekt został nazwany *Floor*.

* Position [ X: 0, Y: -0.5, Z: 0 ]
* Rotation [ X: 0, Y: 0, Z: 0 ]
* Scale [ X: 20, Y: 1, Z: 20 ]

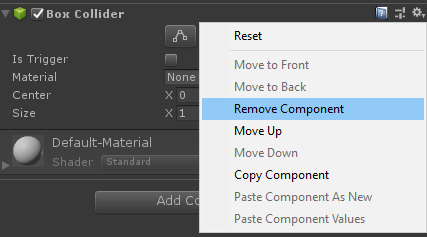
Aby utworzyć kolejne ściany pokoju podłoga została zduplikowana pięciokrotnie zaznaczając obiekt podłogi i klikając na klawiaturze *Ctrl + D* pięć razy. Zduplikowanym obiektom nadane zostały transformacje podane w tabeli.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Position** | | | **Rotation** | | | **Scale** | | |
|  | **X** | **Y** | **Z** | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| **Roof** | 0 | 10.5 | 0 | 0 | 0 | 0 | 20 | 1 | 20 |
| **Front** | 0 | 5 | 10.5 | 90 | 0 | 0 | 20 | 1 | 12 |
| **Back** | 0 | 5 | -10.5 | 90 | 0 | 0 | 20 | 1 | 12 |
| **Left** | -10.5 | 5 | 0 | 90 | 90 | 0 | 20 | 1 | 12 |
| **Right** | 10.5 | 5 | 0 | 90 | 90 | 0 | 20 | 1 | 12 |

Tabela 1. Wartości transformacji ścian pokoju

Czy tutaj powinna być też wartość podłogi?

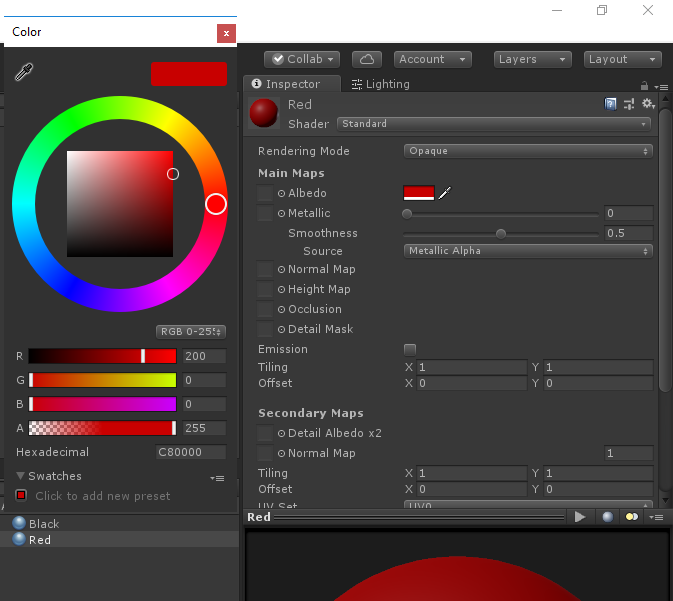
W odróżnieniu od podłogi ściany pokoju nie potrzebują wykrywania kolizji – użytkownik poruszać się będzie w obszarze mniejszym niż ściany. Ze wszystkich obiektów poza *Floor* usunięty został komponent *Box Collider*. W menu które otwiera się po otworzeniu zębatki wybrana została opcja *Remove Component.*



Rysunek . Usuwanie komponentu Box Collider

Następnie w podobny sposób jak ściany, klikając prawym klawiszem myszy i wybierając opcję *Create empty* utworzony pusty obiekt, który został nazwany *Walls*. Obiekt ten także został zresetowany. Wszystkie utworzone ściany zostały przeniesione w hierarchii pod obiekt *Walls* stając się jego częścią.

Kolejną częścią było utworzenie ringu. Aby stworzyć słupki na rogach najpierw w menu otwierającym się po kliknięciu prawym klawiszem myszy należało wybrać opcję *Object 3D > Cyllinder.* Po utworzeniu został on nazwany *Pole1,* transformacja obiektu została zresetowana, a komponent *Capsule Collider* został z niego usunięty. Chcąc zmienić kolor obiektu konieczne było stworzenie nowego materiału. Materiał został utworzony klikając prawym klawiszem myszy w oknie *Project* i wybierając opcję *Create > Material*. Jedyną rzeczą która musiała zostać zmieniona w nowym materiale jest kolor. Po kliknięciu w opcję *Albedo* otwiera się okno, które umożliwia zmianę koloru.



Rysunek . Tworzenie nowego materiału

Aby nadać kolor słupkowi należało przenieść materiał z okna *Project* na obiekt w scenie. Obiekt został trzykrotnie zduplikowany a poszczególnym słupkom nadane zostały opcje transformacji podane w tabeli:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Position** | | | **Rotation** | | | **Scale** | | |
|  | **X** | **Y** | **Z** | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| **Pole1** | -4 | 0.8 | -4 | 0 | 0 | 0 | 0.2 | 0.8 | 0.2 |
| **Pole2** | 4 | 0.8 | -4 | 0 | 0 | 0 | 0.2 | 0.8 | 0.2 |
| **Pole3** | -4 | 0.8 | 4 | 0 | 0 | 0 | 0.2 | 0.8 | 0.2 |
| **Pole4** | 4 | 0.8 | 4 | 0 | 0 | 0 | 0.2 | 0.8 | 0.2 |

Tabela 2. Wartości transformacji słupków

Podobnie jak ściany obiekty zostały zgrupowane w nowym obiekcie o nazwie *Poles.*

Następnym krokiem było utworzenie lin, które znajdują się pomiędzy słupkami. Ponownie użyte tutaj zostały cylindry. Cztery utworzone liny zostały zresetowane, a ich transformacje przedstawione są w tabeli.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Position** | | | **Rotation** | | | **Scale** | | |
|  | **X** | **Y** | **Z** | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| **Line1** | 0 | 0.3 | 0 | 90 | 0 | 0 | 0.05 | 4 | 0.05 |
| **Line2** | 0 | 0.7 | 0 | 90 | 0 | 0 | 0.05 | 4 | 0.05 |
| **Line3** | 0 | 1.1 | 0 | 90 | 0 | 0 | 0.05 | 4 | 0.05 |
| **Line4** | 0 | 0.4 | 0 | 90 | 0 | 0 | 0.05 | 4 | 0.05 |

Tabela 3. Wartości transformacji lin

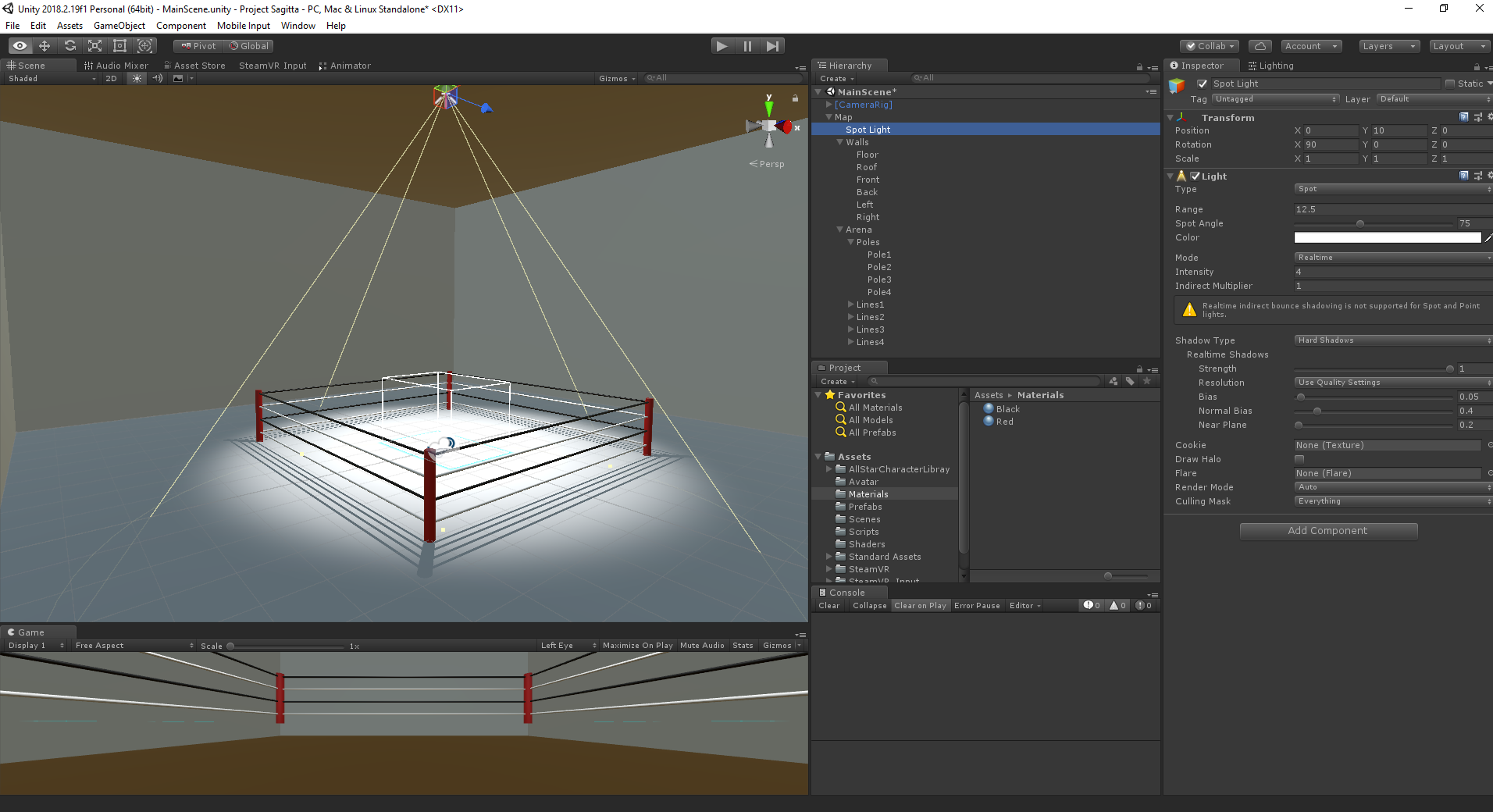
Stworzony został nowy, czarny materiał. Został on nadany obiektom *Line2*i *Line4.* Obiekty zostały zgrupowane do nowego obiektu o nazwie *Lines1.* Obiekt *Lines1* został następnie powielony trzykrotnie, a poszczególnym obiektom nadane zostały wartości transformacji przedstawione w tabeli.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Position** | | | **Rotation** | | | **Scale** | | |
|  | **X** | **Y** | **Z** | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| **Lines1** | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| **Lines2** | -4 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| **Lines3** | 0 | 0 | -4 | 0 | 90 | 0 | 1 | 1 | 1 |
| **Lines4** | 0 | 0 | 4 | 0 | 90 | 0 | 1 | 1 | 1 |

Tabela 4. Wartości transformacji grup lin

Obiekty *Poles, Lines1, Lines2, Lines3* i *Lines4* zostały zgrupowane w obiekcie *Map.*

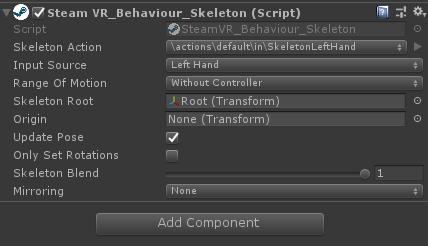
Następnie pod obiektem *Map* zostało utworzone światło typu *Spotlight,* obiekt został umieszczony na górze mapy, aby światło padało na arenę oświetlając ją, resztę pokoju pozostawiając w półmroku.



Rysunek . Wygląd utwrzonej mapy wraz z pokazaną hierarchią obiektów

## 4.3. KONFIGURACJA RUCHU DŁONI

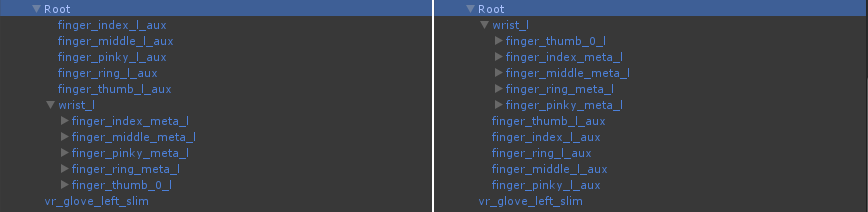
Do projektu zostały dodane modele dłonii, które się w folderze z plikami pluginu SteamVR. Nazwa modeli dłoni które zostały użyte w projekcie to *vr\_glove\_left\_model\_slim* oraz *vr\_glove\_right\_model\_slim*. Do modeli dodane został dodany komponent *Steam VR\_Behaviour\_Skeleton*. Wartości konfiguracyjne ustawione dla lewej dłoni w komponencie przedstawione są na Rys. 15.



Rysunek . Konfiguracja komponentu szkieletu dla lewej dłoni

Wartości dla prawej dłoni są bardzo podobne, różnice polegają w opcji *Skeleton Action* i *Input Source*. Obie te opcje wskazują na wartości *RightHand*.

W dodanych modelach należało także zmienić kolejność punktów transformacji modelu. W oryginalnych rękawicach kości znajdują się w nieodpowiedniej kolejności  
i zgięcia nie odzwierciedlają faktycznego układu dłoni.



Rysunek . Nieprawidłowy (po lewej) i prawidłowy (po prawej) układ kości dłoni w modelu

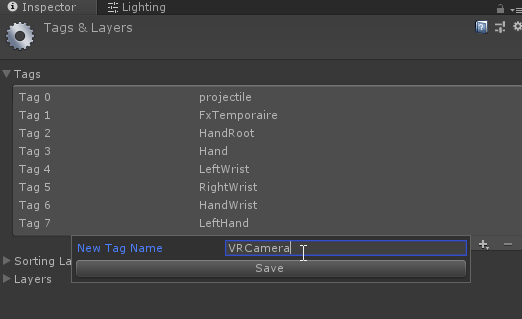
## 4.4. PORUSZANIE POSTACI

### 4.4.1. Przemieszczanie modelu w przestrzeni

Stworzenie odpowiednio przemieszczania w przestrzeni jest wymagane, aby w trakcie rozgrywki drugi użytkownik widział model gracza w sposób bliski temu w którym miejscu i w jakiej pozycji aktualnie znajduje się użytkownik. Przy goglach wirtualnej rzeczywistości należy uwzględniać ruchy głowy oraz ruchy dłoni w przestrzeni świata.

Do projektu zaimportowany został model postaci pobrany z *Unity Asset Store*[1]. Dla modelu utworzony został nowy skrypt o nazwie *PlayerController*. Skrypt posiada 3 zmienne publiczne, które następnie zostały ustawione w edytorze: *head, leftHand*i *rightHand.* W edytorze zmienne te reprezentowane są przez obiekty *Head, L Hand*i *R Hand,* które w strukturze znajdują się pod obiektem *ROOT* modelu. Używane są także zmienne *vrHead, vrLeftHand* i *vrRightHand.*

Przy komunikacji sieciowej, każdy z graczy będzie miał własne obiekty kontrolerów i głowy, przez co zmienne te muszą być zostać odnalezione dynamicznie. Aby było to możliwe potrzebne było utworzenie etykiet dla modeli dłoni. Aby to zrobić należało użyć komendy *Edit > Project Settings > Tags and Layers* w sekcji *Tags* dodane zostały etykiety *LeftWrist, RightWrist* i *VRCamera* (Rys. 18).



Rysunek . Dodawanie etykiet do projektu

Etykiety zostały nadane obiektom *Camera* w *[CameraRig]* oraz obiektom *wrist\_l*  i *wrist\_r* w *ROOT* modeli lewej i prawej dłoni. W skrypcie *PlayerController* utworzone zostały funkcje odpowiedzialne za odszukanie obiektów z odpowiednimi etykietami.

private void FindVRHead() {

GameObject head = GameObject.FindWithTag("VRCamera");

if (head) {

vrHead = head.transform;

}

}

W powyższym kodzie szukany jest obiekt z etykietą *VRCamera* i przypisywany do zmiennej. Podobne funkcje odpowiadają za szukanie lewego oraz prawego modelu dłoni.

Aby zmieniać pozycję dłoni wymagane było przepisanie wartości z pozycji oraz rotacji dłoni w świecie. Należy jednak zauważyć, że z uwagi na to jak skonstruowany został model postaci potrzebna była dodatkowa rotacja. Kod odpowiadający za ruch znajduje się poniżej.

leftHand.transform.position = vrLeftWrist.position;

leftHand.transform.rotation = vrLeftWrist.rotation;

leftHand.transform.Rotate(90f, 0f, -90f);

rightHand.transform.position = vrRightWrist.position;

rightHand.transform.rotation = vrRightWrist.rotation;

rightHand.transform.Rotate(-90f, 180f, -90f);

Rotacja w unity używa zmiennych typu *Quaternion*[2], aby uprościć obliczenia rotacji dla komputera. Aby jednak ułatwić zadanie programistom, kwaterniony można przekształcić na kąty Eulera. Przekształcanie to przydaje się w wielu momentach. Jednym z nich była chęć uniemożliwienia rotacji modelu w sytuacjach, które sprawiałyby, że model wyglądałby źle, np. model głowy skierowany byłby do środka ciała. Do ograniczenia rotacji posłużył kod poniżej, który blokuje za duże obroty głowy względem ciała na osiach X i Z.

transform.position = new Vector3(

vrHead.position.x,

transform.position.y,

vrHead.position.z);

if (head.transform.rotation.eulerAngles.x > 60f && head.transform.rotation.eulerAngles.x < 180f) {

head.transform.rotation = Quaternion.Euler(

60f,

head.transform.rotation.eulerAngles.y,

head.transform.rotation.eulerAngles.z);

}

if (head.transform.rotation.eulerAngles.z > 300f) {

head.transform.rotation = Quaternion.Euler(

head.transform.rotation.eulerAngles.x,

head.transform.rotation.eulerAngles.y,

300f);

} else if (head.transform.rotation.eulerAngles.z < 235f) {

head.transform.rotation = Quaternion.Euler(

head.transform.rotation.eulerAngles.x,

head.transform.rotation.eulerAngles.y,

235f);

}

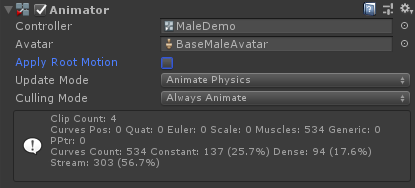
Kod nie zmienia położenia głowy na osi Y, a następnie generuje nowe rotacje jeśli te, które aktualnie są używane są nieodpowiednie. Do utworzenia rotacji typu *Quaternion* na podstawie kątów Eulera służy funkcja *Quaterion.Euler().*

### 4.4.2. Odwrotna kinematyka postaci

Kinematyka odwrotna pozwala na zmianę pozycji elementów będącymi rodzicami przez obiekty będące dziećmi. Przykładem jest tutaj ruch dłoni który zmienia układ rąk i tułowia.

W projekcie przyjęte zostało, że jeśli wartość absolutna różnicy rotacji kamery sterowanej kontrolerem na głowie na osi Y oraz aktualnej rotacji modelu postaci wynosi 30 lub więcej stopni to należy wykonać obrót tej postaci o 45 stopni bliżej do kamery. Po obrocie w stronę kamery wartość absolutna różnicy wynosi ok 15 stopni.

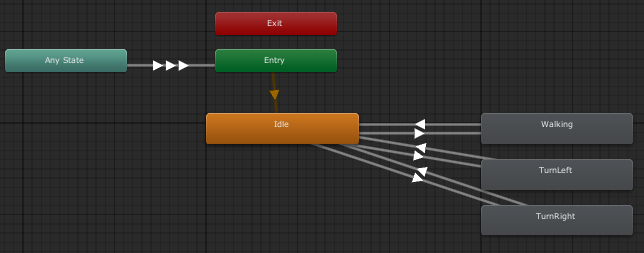
Aby obrót postaci był możliwy z animacjami w obiekcie postaci stworzony skonfigurowany został komponent *Animator*. Odpowiednie obiekty zostały pobrane i zaimportowane wraz z modelem. Konfiguracja przedstawiona jest na (Rys. 18).



Rysunek . Konfiguracja komponentu Animator

Następnie potrzebne było skonfigurowanie animacji. Okno *Animator* zostało uruchomione poprzez wybranie opcji *Window > Animation > Animator.* W karcie *Layers* przy *Base Layer* klikając ikonę zębatki włączono *IK Pass.* Na karcie *Parameters* zostały dodane trzy parametry: *AbsoluteMovement* typu *Float* oraz *TurnLeft* i *TurnRight* typu *Bool.*

Kolejnym krokiem było stworzenie stanów i przejść między nimi. Robi się to przez kliknięcie prawym klawiszem myszy w terminalu i wybranie opcji *Create State > Empty*. Cztery stany potrzebne do poruszania to *Idle, Walking, TurnLeft* i *TurnRight*. Idle jest stanem domyślnym i będzie on wykonywany zawsze, gdy . Zostało to zrobione przez wybranie opcji *Set as Layer Default State*. Poprzez opcję *Make Transition* między stanem *Idle* a *Walking, TurnLeft i TurnRight* zostało utworzona możliwość zmiany stanu.

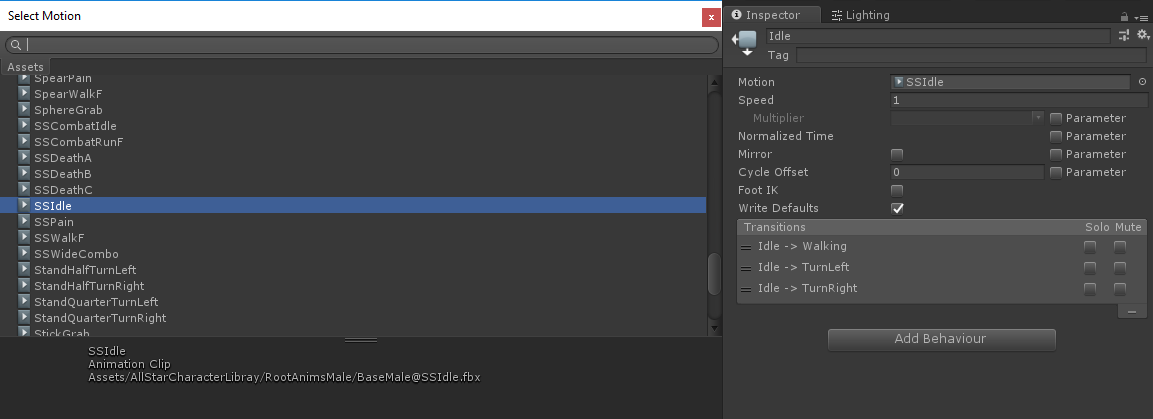


Rysunek . Wygląd maszyny stanów po dodaniu przejść między stanami

Przejście ze stanu *Idle* do *Walking* nie powinno następować przy niewielkich ruchach postaci, ale nie można też pozwolić, żeby użytkownik przemieszczał się całkowicie bez animacji. W sekcji *Conditions* wybrany został parametry *AbsoluteMovement Greater 0.03*. Natomiast przy przejściu ze stanu *Walking* do *Idle* ustawione zostały parametry *AbsoluteMovement Less 0.03.*W obu przejściach odznaczona została też opcja *Has Exit Time*. Opcja ta pozwala na przerwanie animacji jeśli warunek zmienił się przed jej ukończeniem.

W przypadku stanów zmiany ze stanu *Idle* do *TurnLeft* i *TurnRight* w sekcji *Conditions* ustawione zostały parametry *TurnLeft true* i *TurnRight true*. Przy przejściu w drugą stronę warunek został zmieniony na *false.* Tutaj także wyłączona została opcja *Has Exit Time*.

Kolejnym krokiem było skonfigurowanie animacji dla poszczególnych stanów. Pierwszym z nich był stan *Idle*. Dla tego stanu wybrana została animacja *SSIdle* pobrana wraz z modelem. W stanach *Walking, TurnLeft* i *TurnRight* zostały odpowiednio wybrane animacjeo nazwach *SSWalkF*, *StandQuarterTurnLeft* i *StandQuarterTurnRight.*



Rysunek . Wybór animacji dla stanu Idle

Po skonfigurowaniu animacji nastąpiła faza ich implementacji. Do obiektu postaci dodane zostały dwa puste obiekty, które zostały ustawione w miejscu w którą stronę mniej więcej powinny być skierowane łokcie. Utworzony został nowy skrypt nazwie *IKHandling.* Skrypt ten podobnie jak *PlayerController* znajduje obiekty z etykietami *LeftWrist* i *RightWrist*. Na ich podstawie układane są animacje. Kod odpowiadający za zmianę modelu w zależności od położenia dłoni znajduje się poniżej.

animator.SetIKPositionWeight(AvatarIKGoal.LeftHand, ikWeight);

animator.SetIKPosition(AvatarIKGoal.LeftHand, leftHandIKTarget.position);

animator.SetIKRotationWeight(AvatarIKGoal.LeftHand, ikWeight);

animator.SetIKRotation(AvatarIKGoal.LeftHand, leftHandIKTarget.rotation);

animator.SetIKHintPositionWeight(AvatarIKHint.LeftElbow, ikWeight);

animator.SetIKHintPosition(AvatarIKHint.LeftElbow, hintLeftHand.position);

Kod ustawia pozycję, rotację i wskazanie miejsca skierowania łokci oraz wagę poszczególnych elementów. W projekcie wszystkie wagi zostały ustawione na 1. *IKHandling* sprawia, że pozycja rąk zmienia się w zależności od ruchu dłoni.

Kolejnym stworzonym elementem było uwzględnianie ruchów modelu przy ruchu głowy i obrów w konkretną stronę. W *PlayerController* obliczana jest szybkość ruchu i ustawiany jest odpowiedni parametr animatora, co prezentuje kod poniżej.

float absoluteMovement = (Mathf.Abs(transform.position.x - vrHead.position.x) + Mathf.Abs(transform.position.z - vrHead.position.z)) \* Time.deltaTime \* 1000f;

animator.SetFloat("AbsoluteMovement", absoluteMovement);

Następnie należało obliczyć różnicę kątów między rotacją głowy, a rotacją postaci oraz jeśli rotacja ta jest większa niż 30 stopni to podjąć odpowiednią akcję, chyba, że użytkownik jest w trakcie animacji. Podejmowane akcje to blokada animacji, zmianę kąta który chce osiągnąć postać oraz uruchomienie odpowiedniej animacji. Jeśli użytkownik jest w trakcie animacji to powinna nastąpić rotacja postaci w tym kierunku. Implementację powyższych założeń przedstawia kod w listingu.

if (!lockRotation) {

float phi = Mathf.Abs(a - b) % 360f;

float distance = phi > 180f ? 360f - phi : phi;

float signedDistance = distance \* ((a - b >= 0 && a - b <= 180) || (a - b <= -180 && a - b >= -360) ? 1 : -1);

if (signedDistance > 30f) {

lockRotation = true;

StartCoroutine(RunFinishTurnTask());

neededRotation = Quaternion.Euler(0f, a - 45f, 0f);

animator.SetBool("TurnLeft", true);

} else if (signedDistance < -30f) {

lockRotation = true;

StartCoroutine(RunFinishTurnTask());

neededRotation = Quaternion.Euler(0f, a + 45f, 0f);

animator.SetBool("TurnRight", true);

}

} else {

transform.rotation = Quaternion.RotateTowards(transform.rotation, neededRotation, Time.deltaTime \* 180f);

}



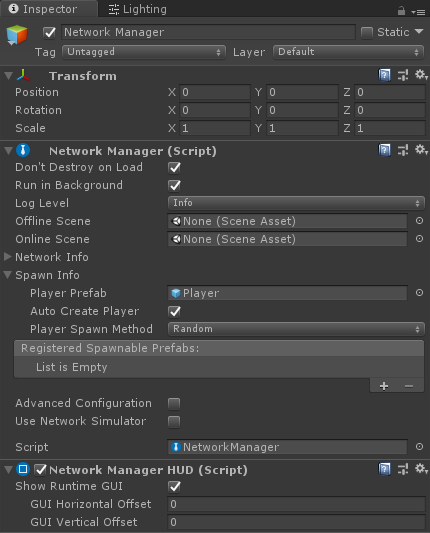
Rysunek . Wygląd postaci po zmianach w odwrotnej kinematyce

## 4.5. KOMUNIKACJA SIECIOWA

### 4.5.1. Synchronizacja ruchu postaci

Przed rozpoczęciem synchronizacji obiekt postaci należało zapisać jako prefab, obiekt, który może być potem dynamicznie utworzony w aplikacji.

Aby synchronizacja przez Internet zadziałała utworzono nowy pusty obiekt, który został nazwany *Network Manager*. Do tego obiektu dodane zostały komponenty, skrypty *Network Manager* i *Network Manager HUD*. Pierwszy zapewnia komunikację sieciową, a drugi prosty interfejs pozwalający połączyć się z serwerem. W *Network Managerze* najważniejsze było ustawienie zmiennej *Player Prefab* na stworzony wcześniej prefab.



Rysunek . Konfiguracja Network Managera

Do stworzonego prefabu dodane teraz zostały komponenty *Network Identity* i *Network Transform*. W pierwszym z nich należy włączyć opcję *Local Player Authority*, a w drugimzmienić *Rotation Axis* na *XYZ (full 3D),* aby zmiany w ruchu obiektu były przekazywane niezależne od osi na której występują.

W skryptach *PlayerController* i *IKHandling* wymagane są zmiany, ponieważ w obecnym stanie skrypty dla obiektów postaci będą się także uruchamiać na komputerach innych graczy. Należy sprawdzić, czy obiekt jest lokalnym obiektem gracza i tylko wtedy uruchomić.

Należy także zauważyć, że w tym momencie obiekty dłoni znajdowały się poza obiektem postaci. Nie jest jednak możliwe przeniesienie ich do tego obiektu, gdyż posiadają one skrypty *Steam VR\_Behaviour\_Skeleton*, które muszą być używane lokalnie, na komputerze użytkownika. Obiekty te zostały jednak skopiowane, a skrypt ten został usunięty. Brak tego skryptu nie pozwala jednak na ruchy dłoni. Dodatkowym skryptem, uruchamianym tylko lokalnie, który został do tego celu utworzony jest *HandMovementCopier*. Pozwala on skopiować dane transformacji wszystkich dzieci obiektu na inny obiekt o identycznej strukturze. Obiekty dłoni do których zostaną przekazane dane transformacji ustalone są w prefabie, a obiekty dłoni kontrolerów ładowane są dynamicznie. Kopiowanie pozycji i rotacji następuje przy każdej ich zmianie i jest przedstawione w kodzie poniżej.

for (int i = 0; i < leftHandNodes.Count; ++i) {

leftHandNodes[i].position = vrLeftHandNodes[i].position;

leftHandNodes[i].rotation = vrLeftHandNodes[i].rotation;

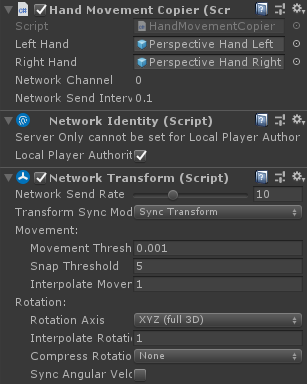
}

for (int i = 0; i < rightHandNodes.Count; ++i) {

rightHandNodes[i].position = vrRightHandNodes[i].position;

rightHandNodes[i].rotation = vrRightHandNodes[i].rotation;

}



Rysunek . Konfiguracja skryptów sieciowych obiektu postaci

### 4.5.2. Stworzenie lobby

## 4.6. SWORZENIE STANU GRY

## 4.7. DODAWANIE DŹWIĘKÓW

# TESTOWANIE

# ZAKOŃCZENIE

## 6.1. PODSUMOWANIE

## 6.2. MOŻLIWOŚCI ROZWOJU

# PRZYPISY

[1] Sci Fi Officer Captain w Unity Asset Store, <https://assetstore.unity.com/packages/3d/characters/sci-fi-officer-captain-52376> (dostęp: 19.11.2018)

[2] Unity – Scripting API: Quaternion, https://docs.unity3d.com/ScriptReference/Quaternion.html (dostęp: 10.01.2019)

# BIBLIOGRAFIA

1. Unity User Manual (2018.2), <https://docs.unity3d.com/2018.2/Documentation/Manual/> (dostęp: 18.11.2018).
2. Oculus Documentation, <https://developer.oculus.com/documentation/> (dostęp: 13.12.2018).
3. OpenVR API Documentation, <https://github.com/ValveSoftware/openvr/wiki/API-Documentation> (dostęp: 18.11.2018).
4. ​Gańko Tomasz, Gra-zabawka dla niemowląt przygotowana z użyciem w Unity 3D, Toruń 2017
5. Ioannou Nikolaos, Multi-Player​ ​Virtual​ ​Reality​ ​Game based​ ​on​ ​Hand​ ​Tracking​ ​Interaction, Chania 2017.
6. Oculus Rift Development Kit 2 Quick Start Guide, 25.07.2014.

# SPIS ILUSTRACJI

[Rysunek 1. Główne okno edytora Unity 5](#_Toc534939106)

[Rysunek 2. Schemat Oculus Rifta 6](#_Toc534939107)

[Rysunek 3. Kontrolery Oculus Touch 7](#_Toc534939108)

[Rysunek 4. Okno przypisywania przycisków do akcji 8](#_Toc534939109)

[Rysunek 5. Okno główne Visual Studio 2018 9](#_Toc534939110)

[Rysunek 6. Schemat warstwy sieciowej w Unity (stworzyć własny schemat) 10](#_Toc534939111)

[Rysunek 7. Ekran tworzenia projektu w Unity 11](#_Toc534939112)

[Rysunek 8. Okno konfiguracji opcji pluginu SteamVR 12](#_Toc534939113)

[Rysunek 9. Konfiguracji Opcji XR 13](#_Toc534939114)

[Rysunek 10. Okno edytora z uruchomioną podstawową sceną 14](#_Toc534939115)

[Rysunek 11. Resetowanie komponentu w Edytorze 14](#_Toc534939116)

[Rysunek 12. Usuwanie komponentu Box Collider 15](#_Toc534939117)

[Rysunek 13. Tworzenie nowego materiału 16](#_Toc534939118)

[Rysunek 14. Wygląd utwrzonej mapy wraz z pokazaną hierarchią obiektów 18](#_Toc534939119)

[Rysunek 15. Konfiguracja komponentu szkieletu dla lewej dłoni 18](#_Toc534939120)

[Rysunek 16. Nieprawidłowy (po lewej) i prawidłowy (po prawej) układ kości dłoni w modelu 19](#_Toc534939121)

[Rysunek 17. Dodawanie etykiet do projektu 20](#_Toc534939122)

[Rysunek 18. Konfiguracja komponentu Animator 21](#_Toc534939123)

[Rysunek 19. Wygląd maszyny stanów po dodaniu przejść między stanami 22](#_Toc534939124)

[Rysunek 20. Wybór animacji dla stanu Idle 23](#_Toc534939125)

[Rysunek 21. Wygląd postaci po zmianach w odwrotnej kinematyce 24](#_Toc534939126)

[Rysunek 22. Konfiguracja Network Managera 25](#_Toc534939127)

[Rysunek 23. Konfiguracja skryptów sieciowych obiektu postaci 26](#_Toc534939128)

# DODATEK A. MODELE UŻYTE W PROJEKCIE